

Seventh DOE ACTS Collection Workshop
Gearing up Scientific Applications for the Petascale Computing Era

ScaLAPACK

Osni Marques

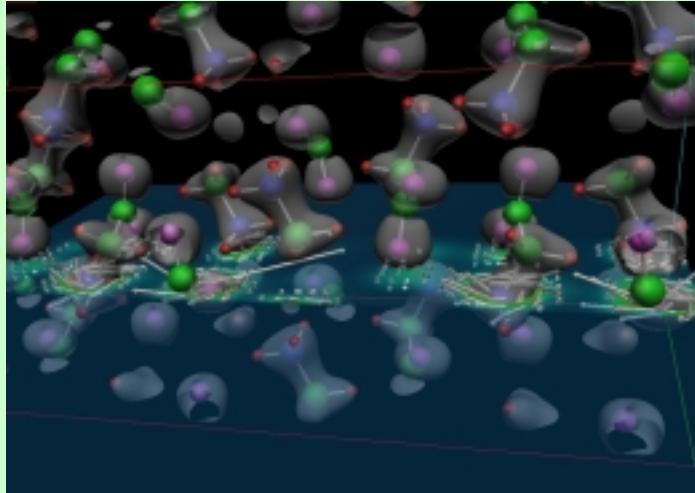
Lawrence Berkeley National Laboratory (LBNL)

OAMarques@lbl.gov

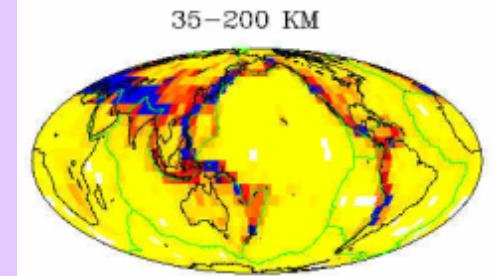
Outline

- Motivation
- ScaLAPACK: *software structure*
 - Basic Linear Algebra Subprograms (BLAS)
 - Linear Algebra PACKage (LAPACK)
 - Basic Linear Algebra Communication Subprograms (BLACS)
 - Parallel BLAS (PBLAS)
- ScaLAPACK: *details*
 - Data layout
 - Array descriptors
 - Error handling
 - Performance
- Hands-on

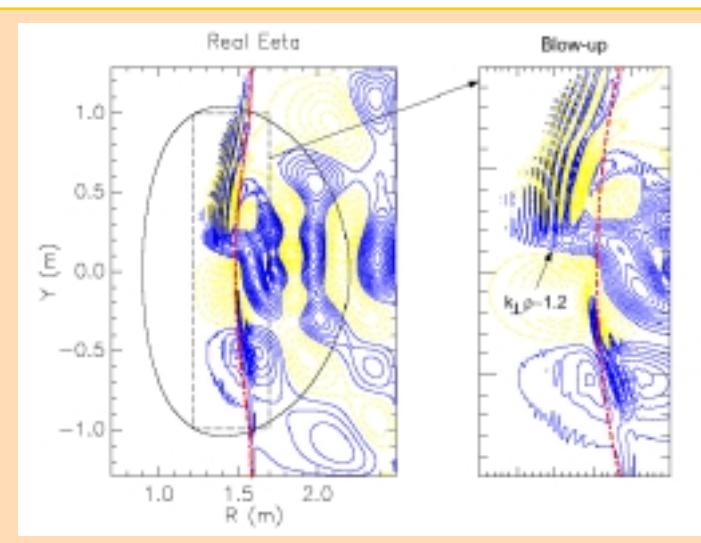
ScaLAPACK: Applications



Induced current (white arrows) and charge density (colored plane and gray surface) in crystallized glycine due to an external field; courtesy of Louie, Yoon, Pfrommer and Canning (UCB and LBNL).



Model for the internal structure of the Earth, resolution matrix (Vasco and Marques)



*Two **ScaLAPACK** routines, PZGETRF and PZGETRS, are used for solution of linear systems in the spectral algorithms based AORSA code (Batchelor et al.), which is intended for the study of electromagnetic wave-plasma interactions. The code reaches 68% of peak performance on 1936 processors of an IBM SP.*

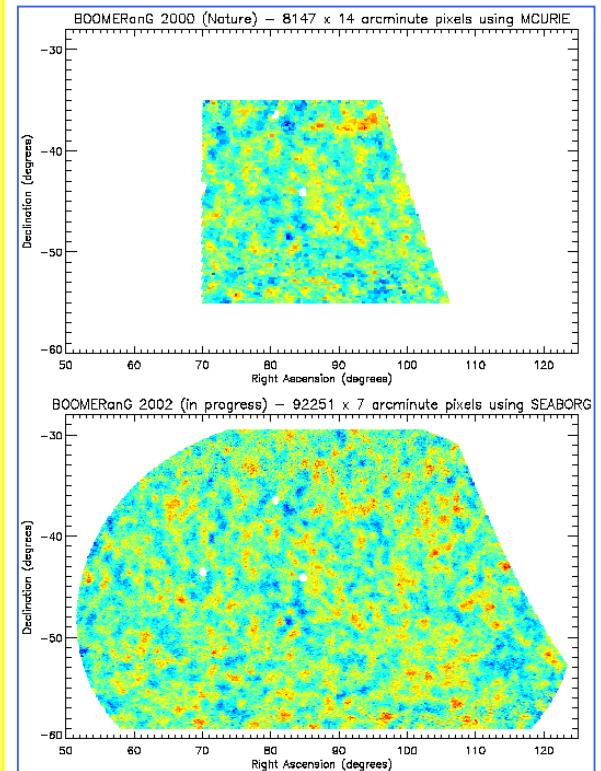
Advanced Computational Research in Fusion (SciDAC Project, PI Mitch Pindzola). Point of contact: Dario Mitnik (Dept. of Physics, Rollins College). Mitnik attended the workshop on the ACTS Collection in September 2000 and afterwards actively used ACTS tools, in particular ScaLAPACK. Dario has worked on the development, testing and support of new scientific simulation codes related to the study of atomic dynamics using time-dependent close coupling lattice and time-independent methods. He has reported that this work could not be carried out in sequential machines and that ScaLAPACK was fundamental for the parallelization of these codes.

Application: Cosmic Microwave Background (CMB) Analysis

- The statistics of the tiny variations in the CMB (the faint echo of the Big Bang) allows the determination of the fundamental parameters of cosmology to the percent level or better.
- MADCAP (Microwave Anisotropy Dataset Computational Analysis Package)
 - Makes maps from observations of the CMB and then calculates their angular power spectra. (See <http://crd.lbl.gov/~borrill>).
 - Calculations are dominated by the solution of linear systems of the form $M = A^{-1}B$ for dense $n \times n$ matrices A and B scaling as $O(n^3)$ in flops. MADCAP uses **ScaLAPACK** for those calculations.
- On the NERSC Cray T3E (original code):
 - Cholesky factorization and triangular solve.
 - Typically reached 70-80% peak performance.
 - Solution of systems with $n \sim 10^4$ using tens of processors.
 - The results demonstrated that the Universe is spatially flat, comprising 70% dark energy, 25% dark matter, and only 5% ordinary matter.
- On the NERSC IBM SP:
 - Porting was trivial but tests showed only 20-30% peak performance.
 - Code rewritten to use triangular matrix inversion and triangular matrix multiplication → one-day work
 - Performance increased to 50-60% peak.
 - Solution of previously intractable systems with $n \sim 10^5$ using hundreds of processors.

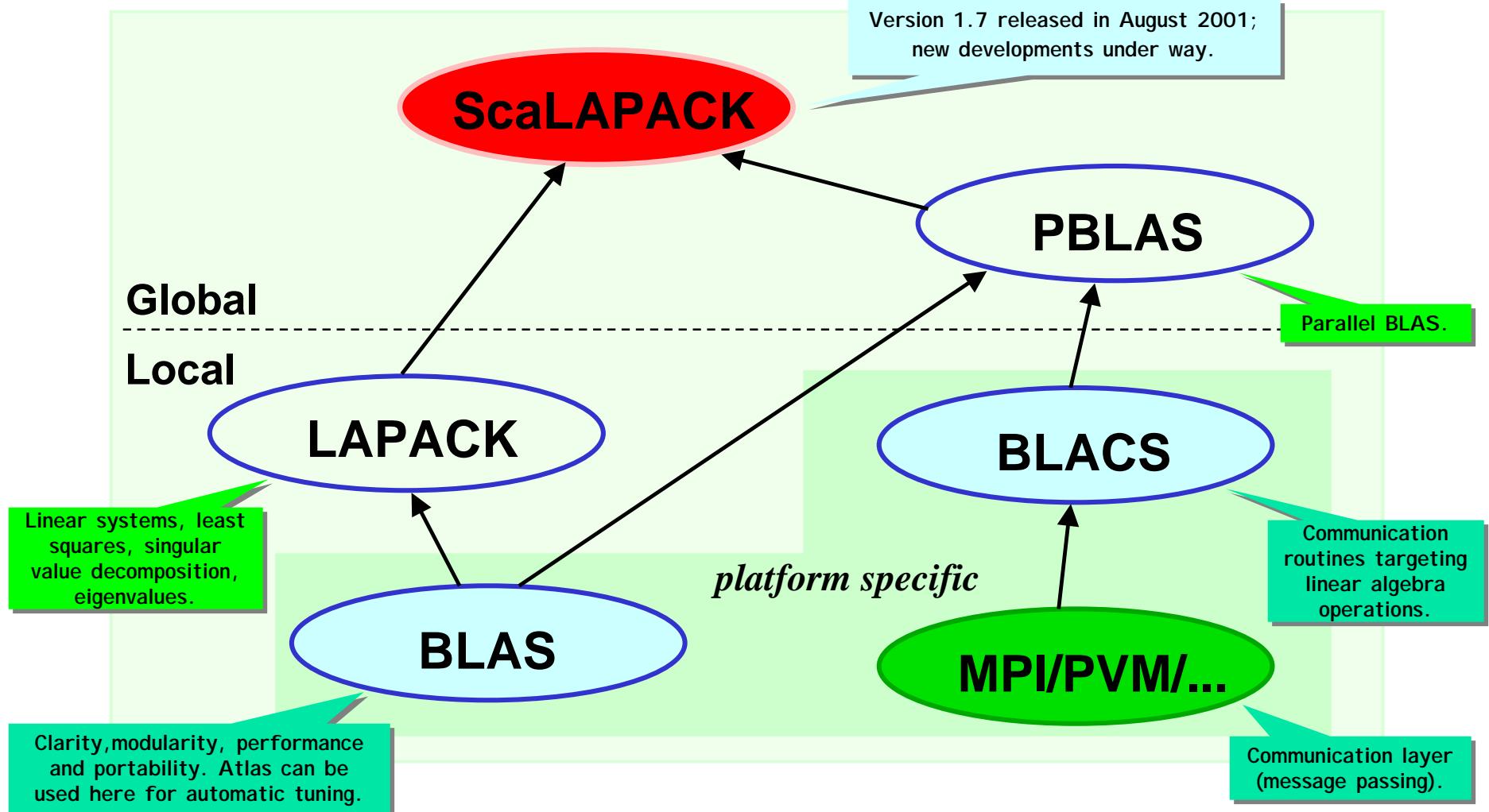


The international BOOMERanG collaboration announced results of the most detailed measurement of the cosmic microwave background radiation (CMB), which strongly indicated that the universe is flat (Apr. 27, 2000).



ScalAPACK: software structure

<http://acts.nersc.gov/scalapack>



BLAS

(*Basic Linear Algebra Subroutines*)

- Clarity: code is shorter and easier to read.
- Modularity: gives programmer larger building blocks.
- Performance: manufacturers (usually) provide tuned machine-specific BLAS.
- Portability: machine dependencies are confined to the BLAS.
- Key to high performance: effective use of memory hierarchy (true on all architectures).

BLAS: 3 levels

- Level 1 BLAS: vector-vector

$$\begin{bmatrix} & \\ & \leftarrow \end{bmatrix} + \begin{bmatrix} & \\ & \square \end{bmatrix} * \begin{bmatrix} & \\ & \end{bmatrix}$$

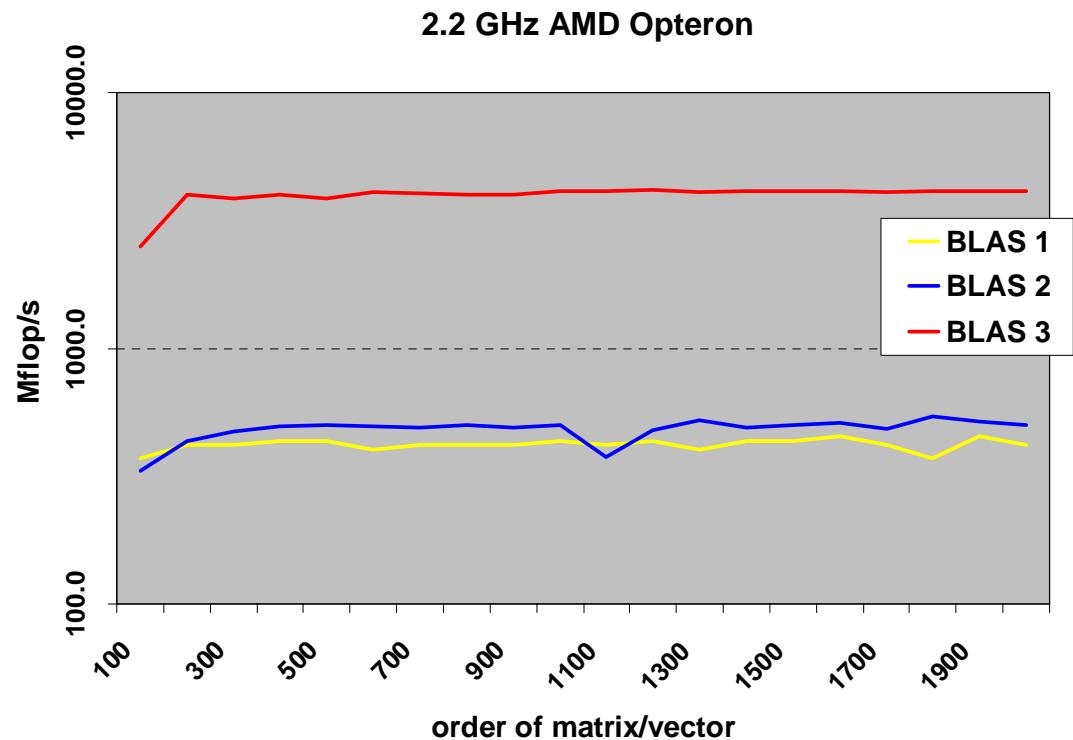
- Level 2 BLAS: matrix-vector

$$\begin{bmatrix} & \\ & \leftarrow \end{bmatrix} \begin{bmatrix} & \\ & \end{bmatrix} * \begin{bmatrix} & \\ & \end{bmatrix}$$

- Level 3 BLAS: matrix-matrix

$$\begin{bmatrix} & \\ & \leftarrow \end{bmatrix} + \begin{bmatrix} & \\ & \end{bmatrix} + \begin{bmatrix} & \\ & \end{bmatrix} * \begin{bmatrix} & \\ & \end{bmatrix}$$

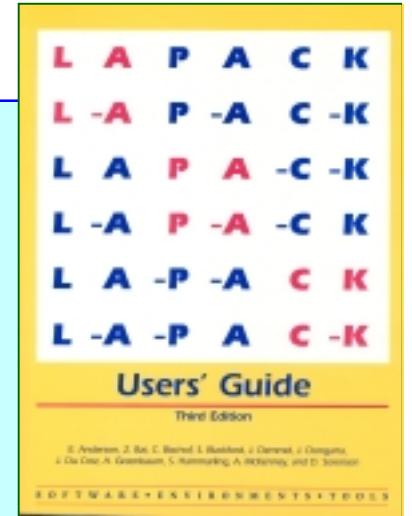
- Clarity
- Portability
- Performance: development of blocked algorithms is important for performance!



LAPACK: *main features*

(<http://www.netlib.org/lapack>)

- Linear Algebra library written in Fortran 77 (Fortran 90)
- Combine algorithms from LINPACK and EISPACK into a single package.
- Efficient on a wide range of computers (RISC, Vector, SMPs).
- Built atop level 1, 2, and 3 BLAS Basic problems:
 - Linear systems: $Ax = b$
 - Least squares: $\min \|Ax - b\|_2$
 - Singular value decomposition: $A = U\Sigma V^T$
 - Eigenvalues and eigenvectors: $Az = \lambda z$, $Az = \lambda Bz$
- LAPACK does not provide routines for structured problems or general sparse matrices (i.e. sparse storage formats such as compressed-row, -column, -diagonal, skyline ...).



BLACS

(*Basic Linear Algebra Communication Subroutines*)

- A design tool, they are a conceptual aid in design and coding.
- Associate widely recognized mnemonic names with communication operations. This improves:
 - program readability
 - self-documenting quality of the code.
- Promote efficiency by identifying frequently occurring operations of linear algebra which can be optimized on various computers.

BLACS: *basics*

- Promote efficiency by identifying common operations of linear algebra that can be optimized on various computers.
- Processes are embedded in a two-dimensional grid.

Example: a 3x4 grid

| | | | | |
|---|---|---|----|----|
| | 0 | 1 | 2 | 3 |
| 0 | 0 | 1 | 2 | 3 |
| 1 | 4 | 5 | 6 | 7 |
| 2 | 8 | 9 | 10 | 11 |

- An operation which involves more than one sender and one receiver is called a *scoped operation*.

| Scope | Meaning |
|--------|------------------------------------------------|
| Row | All processes in a process row participate. |
| Column | All processes in a process column participate. |
| All | All processes in the process grid participate. |

BLACS: *communication routines*

Send/Receive:

`_xxSD2D(ICTXT , [UPLO , DIAG] , M , N , A , LDA , RDEST , CDEST)`
`_xxRV2D(ICTXT , [UPLO , DIAG] , M , N , A , LDA , RSRC , CSRC)`

| _ (Data type) | xx (Matrix type) |
|--------------------------------------------------------------------------------------|----------------------------------------------------------|
| I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex. | GE: General rectangular matrix TR: Trapezoidal matrix |

Broadcast:

`_xxBS2D(ICTXT , SCOPE , TOP , [UPLO , DIAG] , M , N , A , LDA)`
`_xxBR2D(ICTXT , SCOPE , TOP , [UPLO , DIAG] , M , N , A , LDA , RSRC , CSRC)`

| SCOPE | TOP |
|--------------|-------------------|
| 'Row' | ' ' (default) |
| 'Column' | 'Increasing Ring' |
| 'All' | '1-tree' ... |

BLACS: example

```

:
* Get system information
  CALL BLACS_PINFO( IAM, NPROCS )
:
* Get default system context
  CALL BLACS_GET( 0, 0, ICTXT )
:
* Define 1 x (NPROCS/2+1) process grid
  NPROW = 1
  NPCOL = NPROCS / 2 + 1
  CALL BLACS_GRIDINIT( ICTXT, 'Row', NPROW, NPCOL )
  CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
:
* If I'm not in the grid, go to end of program
  IF( MYROW.NE.-1 ) THEN
    IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
      CALL DGESD2D( ICTXT, 5, 1, X, 5, 1, 0 )
    ELSE IF( MYROW.EQ.1 .AND. MYCOL.EQ.0 ) THEN
      CALL DGERV2D( ICTXT, 5, 1, Y, 5, 0, 0 )
    END IF
    :
    CALL BLACS_GRIDEXIT( ICTXT ) ← leave context
  END IF
  :
  CALL BLACS_EXIT( 0 ) ← exit from the BLACS
END

```

(out) uniquely identifies each process
 (out) number of processes available
 (in) integer handle indicating the context
 (in) use (default) system context
 (out) BLACS context
 (output)
 process row and column coordinate
 send X to process (1,0)
 receive X from process (0,0)

- The BLACS context is the BLACS mechanism for partitioning communication space.
- A message in a context cannot be sent or received in another context.
- The context allows the user to
 - create arbitrary groups of processes
 - create multiple overlapping and/or disjoint grids
 - isolate each process grid so that grids do not interfere with each other
- BLACS context \Leftrightarrow MPI communicator

See <http://www.netlib.org/blacs> for more information.

PBLAS

(*Parallel Basic Linear Algebra Subroutines*)

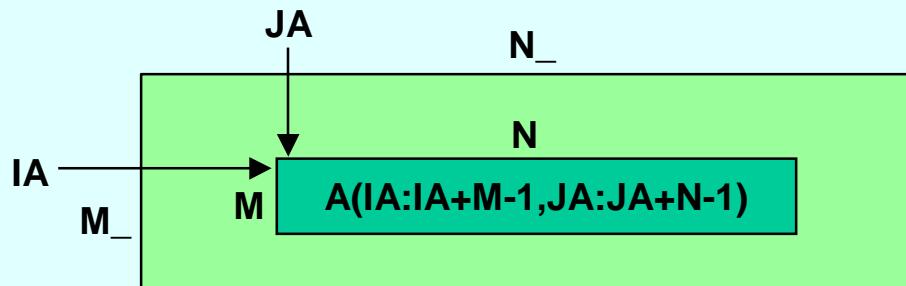
- Similar to the BLAS in portability, functionality and naming.
- Built atop the BLAS and BLACS
- Provide global view of matrix

```
CALL DGEXXX( M, N, A( IA, JA ), LDA, ... )
```

BLAS

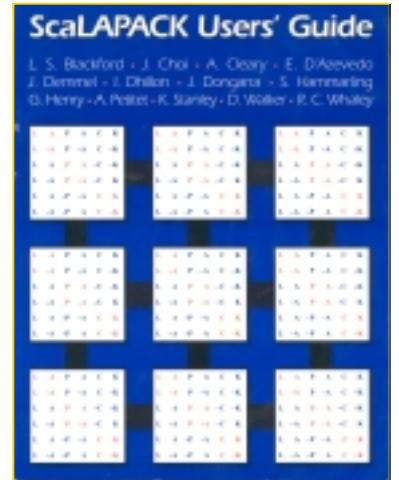
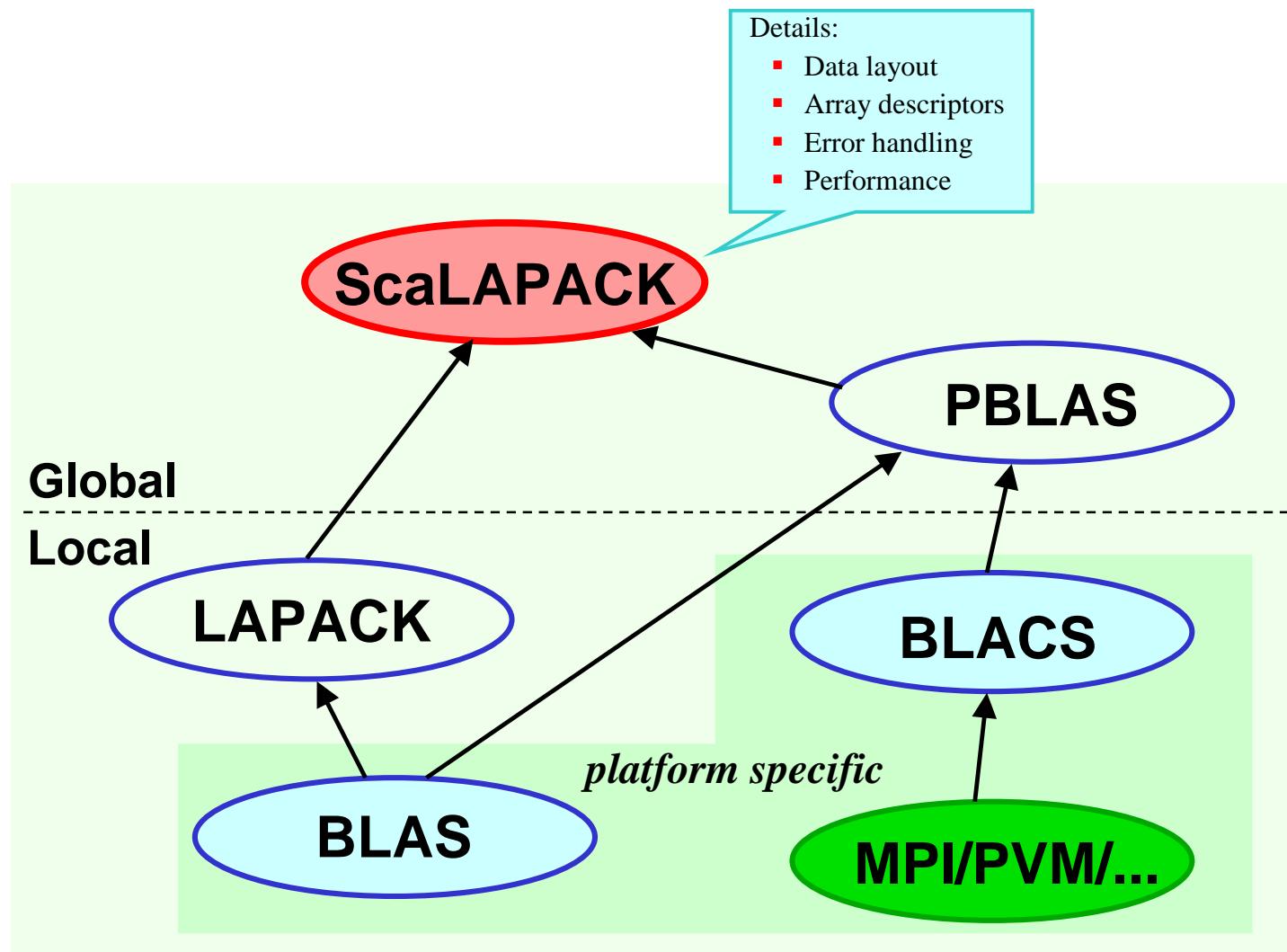
```
CALL PDGEXXX( M, N, A, IA, JA, DESCA, ... )
```

PBLAS



Array descriptor
(see next slides)

ScaLAPACK: structure of the software

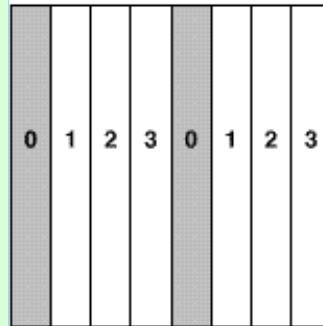
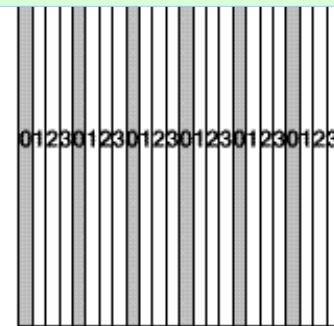
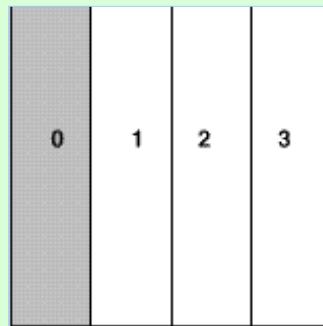


ScaLAPACK: *goals*

- Efficiency
 - Optimized computation and communication engines
 - Block-partitioned algorithms (Level 3 BLAS) for good node performance
- Reliability
 - Whenever possible, use LAPACK algorithms and error bounds.
- Scalability
 - As the problem size and number of processors grow
 - Replace LAPACK algorithm that did not scale (new ones into LAPACK)
- Portability
 - Isolate machine dependencies to BLAS and the BLACS
- Flexibility
 - Modularity: build rich set of linear algebra tools (BLAS, BLACS, PBLAS)
- Ease-of-Use
 - Calling interface similar to LAPACK

ScalAPACK: *data layouts*

- 1D block and cyclic column distributions



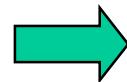
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 |

- 1D block-cycle column and 2D block-cyclic distribution
- 2D block-cyclic used in ScalAPACK for dense matrices

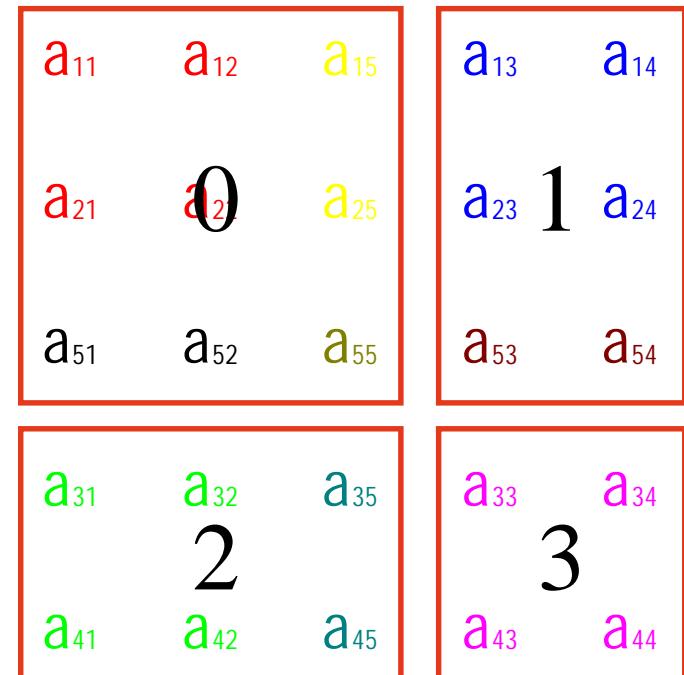
ScalAPACK: 2D Block-Cyclic Distribution

5x5 matrix partitioned in 2x2 blocks

$$\left(\begin{array}{cc|cc|c} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ \hline a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{array} \right)$$



2x2 process grid point of view



2D Block-Cyclic Distribution

| | | | | |
|------|------|------|------|-----|
| 1.1 | 1.2 | 1.3 | 1.4 | 1.5 |
| -2.1 | 2.2 | 2.3 | 2.4 | 2.5 |
| -3.1 | -3.2 | 3.3 | 3.4 | 3.5 |
| -4.1 | -4.2 | -4.3 | 4.4 | 4.5 |
| -5.1 | -5.2 | -5.3 | -5.4 | 5.5 |

```
:
CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
```

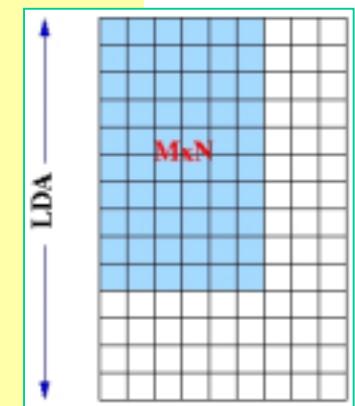
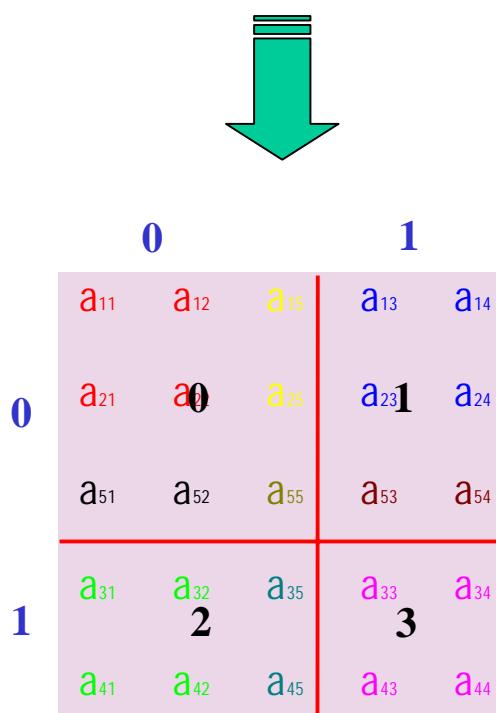
```
IF      ( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
    A(1) = 1.1; A(2) = -2.1; A(3) = -5.1;
    A(1+LDA) = 1.2; A(2+LDA) = 2.2; A(3+LDA) = -5.2;
    A(1+2*LDA) = 1.5; A(2+3*LDA) = 2.5; A(3+4*LDA) = -5.5;
ELSE IF ( MYROW.EQ.0 .AND. MYCOL.EQ.1 ) THEN
    A(1) = 1.3; A(2) = 2.3; A(3) = -5.3;
    A(1+LDA) = 1.4; A(2+LDA) = 2.4; A(3+LDA) = -5.4;
ELSE IF ( MYROW.EQ.1 .AND. MYCOL.EQ.0 ) THEN
    A(1) = -3.1; A(2) = -4.1;
    A(1+LDA) = -3.2; A(2+LDA) = -4.2;
    A(1+2*LDA) = 3.5; A(2+3*LDA) = 4.5;
ELSE IF ( MYROW.EQ.1 .AND. MYCOL.EQ.1 ) THEN
    A(1) = 3.3; A(2) = -4.3;
    A(1+LDA) = 3.4; A(2+LDA) = 4.4;
END IF
```

```
:
```

```
CALL PDGESVD( JOBU, JOBVT, M, N, A, IA, JA, DESC A, S, U, IU,
              JU, DESC U, VT, IVT, JVT, DESC VT, WORK, LWORK,
              INFO )
```

```
:
```

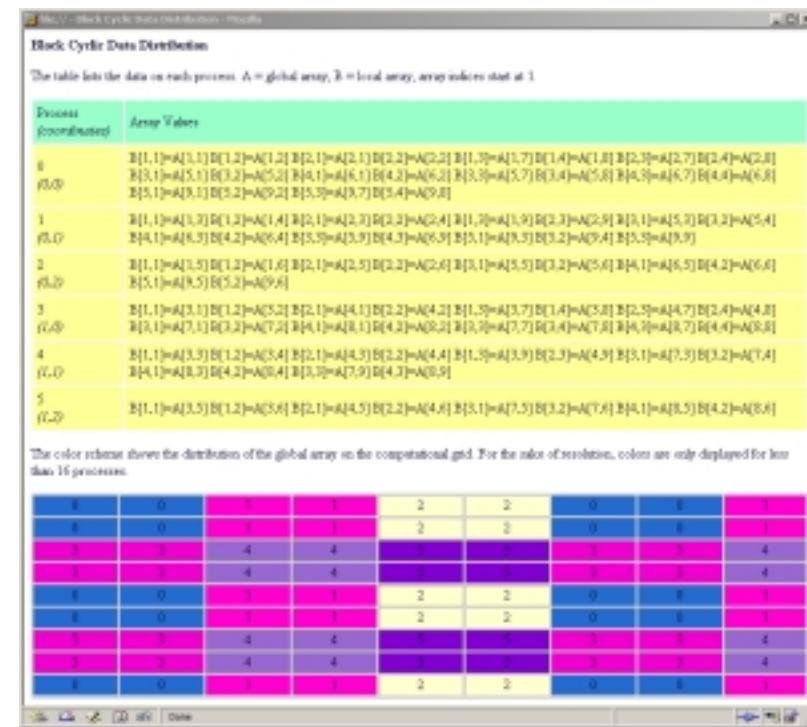
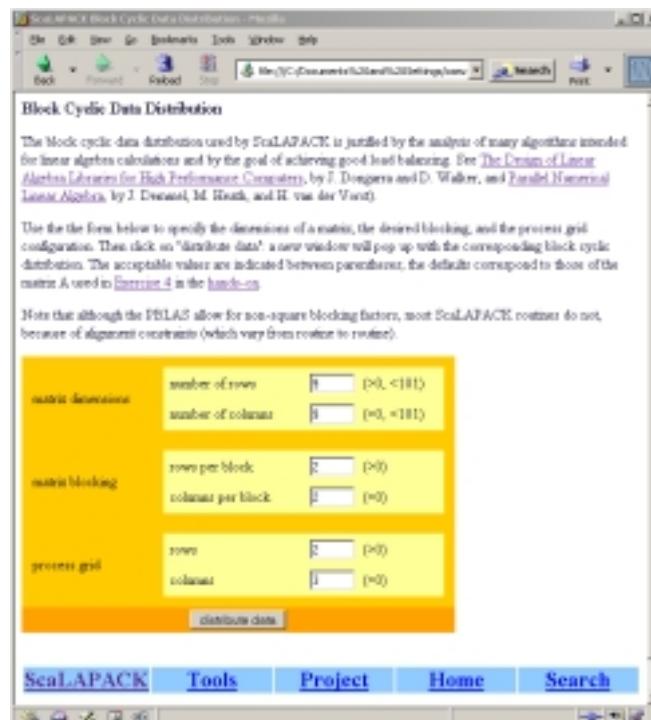
LDA is the leading dimension of the local array (see slides 23-26)



Array descriptor for A
(see slides 23-26)

2D Block-Cyclic Distribution

- Ensures good load balance → performance and scalability (analysis of many algorithms to justify this layout).
- Encompasses a large number of data distribution schemes (but not all).
- Needs redistribution routines to go from one distribution to the other.
- See <http://acts.nersc.gov/scalapack/hands-on/datadist.html>



ScaLAPACK: *array descriptors*

- Each global data object is assigned an *array descriptor*.
- The *array descriptor*:
 - Contains information required to establish mapping between a global array entry and its corresponding process and memory location (uses concept of BLACS context).
 - Is differentiated by the DTYPEn_ (first entry) in the descriptor.
 - Provides a flexible framework to easily specify additional data distributions or matrix types.
- User must distribute all global arrays prior to the invocation of a ScaLAPACK routine, for example:
 - Each process generates its own submatrix.
 - One processor reads the matrix from a file and send pieces to other processors (may require message-passing for this).

Array Descriptor for Dense Matrices

| DESC_0 | Symbolic Name | Scope | Definition |
|--------|---------------|----------|---------------------------------------------------------------------------|
| 1 | DTYPE_A | (global) | Descriptor type DTYPE_A=1 for dense matrices. |
| 2 | CTXT_A | (global) | BLACS context handle. |
| 3 | M_A | (global) | Number of rows in global array A. |
| 4 | N_A | (global) | Number of columns in global array A. |
| 5 | MB_A | (global) | Blocking factor used to distribute the rows of array A. |
| 6 | NB_A | (global) | Blocking factor used to distribute the columns of array A. |
| 7 | RSRC_A | (global) | Process row over which the first row of the array A is distributed. |
| 8 | CSRC_A | (global) | Process column over which the first column of the array A is distributed. |
| 9 | LLD_A | (local) | Leading dimension of the local array. |

Array Descriptor for Narrow Band Matrices

| DESC_() | Symbolic Name | Scope | Definition |
|---------|---------------|----------|-----------------------------------------------------------------------------------------------------------------------------|
| 1 | DTYPE_A | (global) | Descriptor type DTYPE_A=501 for 1 x P _c process grid for band and tridiagonal matrices block-column distributed. |
| 2 | CTXT_A | (global) | BLACS context handle. |
| 3 | N_A | (global) | Number of columns in global array A. |
| 4 | NB_A | (global) | Blocking factor used to distribute the columns of array A. |
| 5 | CSRC_A | (global) | Process column over which the first column of the array A is distributed. |
| 6 | LLD_A | (local) | Leading dimension of the local array. For the tridiagonal subroutines, this entry is ignored. |
| 7 | — | — | Unused, reserved. |

Array Descriptor for Right Hand Sides for Narrow Band Linear Solvers

| DESC_() | Symbolic Name | Scope | Definition |
|---------|---------------|----------|----------------------------------------------------------------------------------------------|
| 1 | DTYPE_B | (global) | Descriptor type DTYPE_B=502 for Pr x 1 process grid for block-row distributed matrices |
| 2 | CTXT_B | (global) | BLACS context handle |
| 3 | M_B | (global) | Number of rows in global array B |
| 4 | MB_B | (global) | Blocking factor used to distribute the rows of array B |
| 5 | RSRC_B | (global) | Process row over which the first row of the array B is distributed |
| 6 | LLD_B | (local) | Leading dimension of the local array. For the tridiagonal subroutines, this entry is ignored |
| 7 | — | — | Unused, reserved |

ScaLAPACK: Functionality

| $Ax = b$ | Simple Driver | Expert Driver | Factor | Solve | Inversion | Conditioning Estimator | Iterative Refinement |
|---------------------------------------|---------------|---------------|-----------|----------|-----------|------------------------|----------------------|
| Triangular | | | | X | X | X | X |
| SPD | X | X | X | X | X | X | X |
| SPD Banded | X | | X | X | | | |
| SPD Tridiagonal | X | | X | X | | | |
| General | X | X | X | X | X | X | X |
| General Banded | X | | X | X | | | |
| General Tridiagonal | X | | X | X | | | |
| Least Squares | X | | X | X | | | |
| GQR | | | X | | | | |
| GRQ | | | X | | | | |
| $Ax = \lambda x$ or $Ax = \lambda Bx$ | Simple Driver | Expert Driver | Reduction | Solution | | | |
| Symmetric | X | X | X | X | | | |
| General | X | X | X | X | | | |
| Generalized BSPD | X | | X | X | | | |
| SVD | | | X | X | | | |

ScaLAPACK: *error handling*

- Driver and computational routines perform *global* and *local* input error-checking.
 - Global checking → synchronization
 - Local checking → validity
- No input error-checking is performed on the auxiliary routines.
- If an error is detected in a PBLAS or BLACS routine program execution stops.

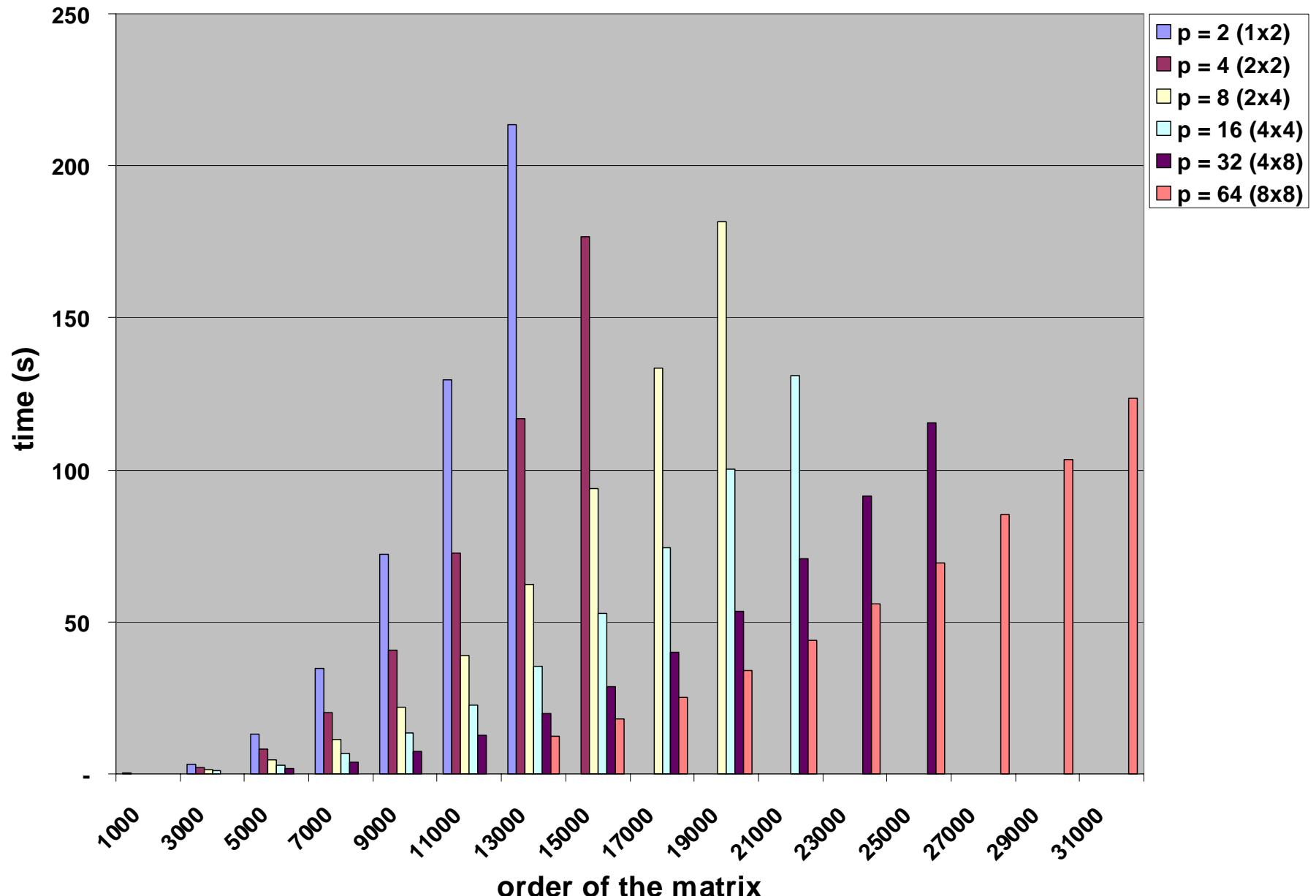
ScaLAPACK: *debugging hints*

- Look at ScaLAPACK example programs.
- Always check the value of INFO on exit from a ScaLAPACK routine.
- Query for size of workspace, LWORK = -1.
- Link to the Debug Level 1 BLACS (specified by BLACSDBGLVL=1 in Bmake.inc).
- Consult errata files on *netlib*:
<http://www.netlib.org/scalapack/errata.scalapack>
<http://www.netlib.org/blacs/errata.blacs>

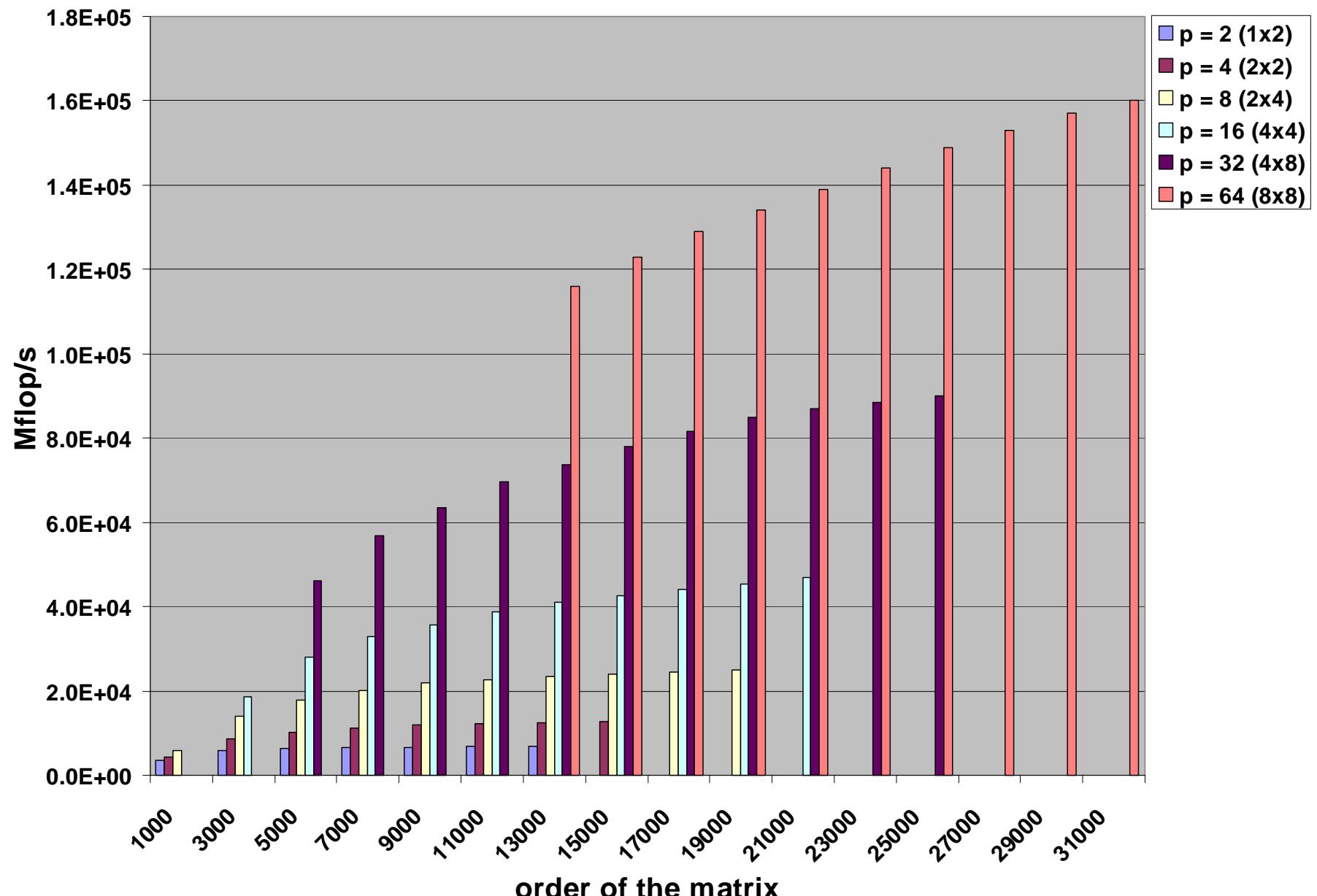
ScaLAPACK: Performance

- The algorithms implemented in ScaLAPACK are scalable in the sense that the parallel efficiency is an increasing function of N^2/P (problem size per node).
- Maintaining memory use per node constant allows efficiency to be maintained (in practice, a slight degradation is acceptable).
- Use efficient machine-specific BLAS (not the Fortran 77 source code available in <http://www.netlib.gov>) and BLACS (nondebug installation).
- On a distributed-memory computer:
 - Use the right number of processors
 - Rule of thumb: $P=MxN/10^6$ for an MxN matrix, which provides a local matrix of size approximately 1000-by-1000.
 - Do not try to solve a small problem on too many processors.
 - Do not exceed the physical memory.
 - Use an efficient data distribution.
 - Block size (i.e., MB,NB) = 64.
 - Square processor grid: Prow = Pcolumn.

LU on 2.2 GHz AMD Opteron (4.4 GFlop/s peak performance)

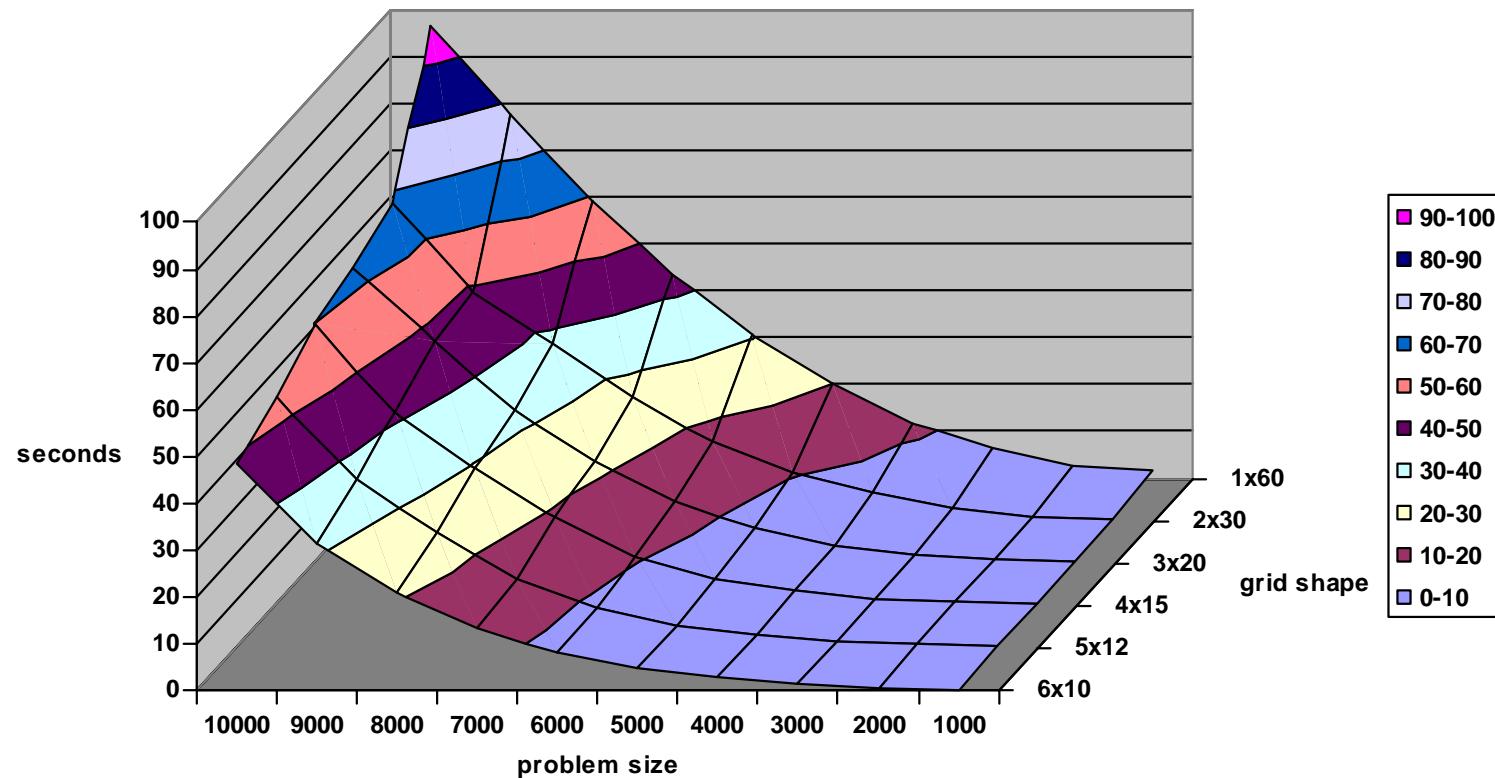


LU+solve on 2.2 GHz AMD Opteron (4.4 GFlop/s peak performance)



ScaLAPACK: *Grid Size Effects*

Execution time of PDGESV for various grid shape



60 processors, Dual AMD Opteron 1.4GHz Cluster with Myrinet Interconnect, 2GB Memory

ScaLAPACK: *Commercial Use*

ScaLAPACK has been incorporated in the following commercial packages:

- Fujitsu
- Hewlett-Packard
- Hitachi
- IBM Parallel ESSL
- NAG Numerical Library
- Cray LIBSCI
- NEC Scientific Software Library
- Sun Scientific Software Library
- Visual Numerics (IMSL)

ScaLAPACK: *Development Team*

- Susan Blackford, UTK
- Jaeyoung Choi, Soongsil University
- Andy Cleary, LLNL
- Ed D'Azevedo, ORNL
- Jim Demmel, UCB
- Inderjit Dhillon, UT Austin
- Jack Dongarra, UTK
- Ray Fellers, LLNL
- Sven Hammarling, NAG
- Greg Henry, Intel
- Sherry Li, LBNL
- Osni Marques, LBNL
- Caroline Papadopoulos, UCSD
- Antoine Petitet, UTK
- Ken Stanley, UCB
- Francoise Tisseur, Manchester
- David Walker, Cardiff
- Clint Whaley, UTK
- Julien Langou, UTK
- :

ScaLAPACK: *Summary*

- Library of high performance dense linear algebra routines for distributed-memory computing
- Reliability, scalable and portable
- Calling interface similar to LAPACK
- New developments on the way
- Online survey: <http://icl.cs.utk.edu/lapack-forum/survey>

Hands-on: <http://acts.nersc.gov/scalapack/hands-on>

Hands-On Exercises for ScaLAPACK

The ScaLAPACK Team

August 2005

Introduction

These exercises provide basic and more advanced programming instruction for writing parallel programs calling the BLACS, PBLAS, and ScaLAPACK. A basic knowledge of Fortran, parallel programming with message-passing, and MPI is assumed. Some of the exercises also require an understanding of two-dimensional block cyclic data distribution.

Exercises 1 and 2 give an introduction to parallel programming with the Basic Linear Algebra Communication Subprograms (BLACS). Exercises 3, 4, and 5 provide a range of simplistic to more complex parallel programs calling PBLAS and ScaLAPACK.

The instructions for the exercises assume that the underlying system is an IBM SP; using up to six processes that do message-passing. These example programs use MPI as the underlying message-passing layer. These hands-on exercises were prepared in collaboration with the University of Tennessee, based on contributions from A. YarKhan, C. Hastings, S. Blackford, C. Whaley, A. Petitet and O. Marques.

Exercise 1: [BLACS - Hello World Example](#)

Exercise 2: [BLACS - Pi Example](#)

Exercise 3: [PBLAS Example](#)

Exercise 4: [ScaLAPACK - Example Program 1](#)

Exercise 5: [ScaLAPACK - Example Program 2](#)

Addition Resources:

- [Block Cyclic Data Distribution](#)
- [Useful calling sequences](#)
- Detailed information on the [BLACS](#)
- Detailed information on the [PBLAS](#)
- Detailed information on [ScaLAPACK](#) and more [examples](#)
- [Download all exercises](#)

[ScalAPACK](#)

[Tools](#)

[Project](#)

[Home](#)

[Search](#)

Hands-on: *instructions*

- Do a “`cp -r /usr/common/acts/SCALAPACK/hands-on hands-on`”.
- There are six subdirectories under *hands-on*:
 - Example 1: BLACS, “hello world” example
 - Example 2: BLACS, “pi” example
 - Example 3: PBLAS example
 - Example 4: ScaLAPACK small problem 1
 - Example 5: ScaLAPACK small problem 2
 - various
- Examples 1-5 are written in Fortran. For a successful compilation and execution, you may have to correct some lines in the examples, in particular the lines starting with ******* (commented lines).
- Examples 1-5 can be compiled with “*make*”, which will generate an executable file with “*.x*”.
- Try also <http://acts.nersc.gov/scalapack/hands-on/datadist.html> with a bigger matrix and different block/grid sizes.

Exercise 4: ScalAPACK - Example Program 1

Information: This is a very simple program that solves a linear system by calling the ScalAPACK routine `PSGESV`. More complete details of this example program can be found in Chapter 2 of the [ScalAPACK Users' Guide](#). This example program demonstrates the basic requirements to call a ScalAPACK routine: initializing the process grid, assigning the matrix to the processes, calling the ScalAPACK routine, and releasing the process grid.

This example program solves the 9-by-9 system of linear equations given by:

```

/ 19  2  1  12  1  16  1  2  11 / / x1 / / 0 /
\ -19  0  1  12  1  16  1  2  11 / \ x2 / \ 0 /
\ -19 -3  1  12  1  16  1  2  11 / \ x3 / \ 3 /
\ -19 -2 -1  12  1  16  1  2  11 / \ x4 / \ 0 /
\ -19 -3 -1 -12  1  16  1  2  11 / * \ x5 / = 0 /
\ -19 -3 -1 -12 -1  16  1  2  11 / \ x6 / \ 0 /
\ -19 -3 -1 -12 -1 -16  1  2  11 / \ x7 / \ 0 /
\ -19 -3 -1 -12 -1 -16 -1  2  11 / \ x8 / \ 0 /
\ -19 -3 -1 -12 -1 -16 -1 -2  11 / \ x9 / \ 0 /

```

using the ScalAPACK driver routine `PSGESV`. The ScalAPACK routine `PSGESV` solves a system of linear equations $A \cdot x = B$, where the coefficient matrix (denoted by A) and the right-hand-side matrix (denoted by B) are real, general distributed matrices. The coefficient matrix A is distributed as depicted below and, for simplicity, we shall solve the system for one right-hand side ($NRHS=1$), that is, the matrix B is a vector. The third element of the matrix B is equal to 1, and all other elements are equal to 0. After solving this system of equations, the solution vector x is given by

```

/ x1 / / 0 /
\ x2 / \ -1/6 /
\ x3 / \ 1/2 /
\ x4 / \ 0 /
\ x5 / \ 0 /
\ x6 / \ 0 /
\ x7 / \ -1/2 /
\ x8 / \ 1/6 /
\ x9 / \ 0 /

```

Let us assume that the matrix A is partitioned and distributed such that we have chosen the row and column block sizes as $MB=NB=2$, and the matrix is distributed on a 2-by-3 process grid ($P_r=2$, $P_c=3$). The partitioning and distribution of our example matrix A is represented in the two figures below, where, to aid visualization, we use the notation $s=19$, $c=3$, $a=1$, $b=12$, $p=16$, and $k=11$.

Partitioning of global matrix A :

```

#   c | a  1 | a  p | a  c | k
#   c | a  1 | a  p | a  c | k
-----+-----+-----+
# -c | a  1 | a  p | a  c | k
# -c | -a  1 | a  p | a  c | k
-----+-----+-----+
# -c | -a -1 | a  p | a  c | k
# -c | -a -1 | -a  p | a  c | k
-----+-----+-----+
# -c | -a -1 | -a -p | -a -c | k
#   c | -a -1 | -a -p | -a -c | k
-----+-----+-----+
# -c | -a -1 | -a -p | -a -c | k

```

PROGRAM PSSCAEX

```

* This program solves a linear system by calling the ScalAPACK
* routine PSGESV. The input matrix and right-and-sides are
* read from a file. The solution is written to a file.

* .. Parameters ..

* ..
* .. Local Scalars ..

* ..
* .. Local Arrays ..

* .. Executable Statements ..

* ..
* .. Executable Statements ..

* Get starting information

*** CALL BLACS_*****(&IAM, NPROCS)
CALL PSSCAEXINFO( OUTFILE, NOUT, N, NRHS, NB, NPROW, NPCOL, MEM,
$ IAM, NPROCS )

* Define process grid
*
*** CALL BLACS_**(-1, 0, ICTXT)
*** CALL BLACS_*****(&ICTXT, 'Row-major', NPROW, NPCOL )
CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )

:
* Read from file and distribute matrices A and B
*
CALL PSLAREAD( 'SCAEXMAT.dat', MEM( IPA ), DESCRA, 0, 0,
$ MEM( IPW ) )
*** CALL *****(&'SCAEXRHS.dat', MEM( IPB ), DESCBC, 0, 0,
$ MEM( IPW ) )

:
*** CALL *****(&N, NRHS, MEM( IPA ), 1, 1, DESCRA, MEM( IPPIV ),
$ MEM( IPB ), 1, 1, DESCBC, INFO )

:

```

line to be
fixed!

lines to
be fixed!

Contents of hands-on/variou

```
[/usr/common/acts/SCALAPACK/hands-on/variou] ls -ls
total 184
8 -rw----- 1 osni acts      2906 Aug 17 16:22 A.SVD
8 -rw----- 1 osni acts      2700 Aug 17 16:22 A.dat
8 -rw----- 1 osni acts      1592 Aug 17 16:22 README
8 -rw----- 1 osni acts      1767 Aug 17 16:22 dcomplex.h
8 -rw----- 1 osni acts      373  Aug 17 16:22 desc.h
8 -rw----- 1 osni acts      4785 Aug 17 16:22 example1.f
8 -rw----- 1 osni acts      6045 Aug 17 16:22 example2.f
16 -rw----- 1 osni acts     10039 Aug 17 16:22 example3.f
8 -rw----- 1 osni acts      3953 Aug 17 16:22 pdtttrdrv.c
8 -rw----- 1 osni acts      3705 Aug 17 16:22 pdtttrdrv.f
8 -rw----- 1 osni acts      365  Aug 17 16:22 pdtttrdrv.i1
8 -rw----- 1 osni acts      13   Aug 17 16:22 pdgesvddrv.dat
8 -rw----- 1 osni acts      6951 Aug 17 16:22 pdgesvddrv.f
8 -rw----- 1 osni acts      389  Aug 17 16:22 pdgesvddrv.i1_3
8 -rw----- 1 osni acts      369  Aug 17 16:22 pdgesvddrv.i1_4
8 -rw----- 1 osni acts      6160 Aug 17 16:22 pdpttr_2.c
8 -rw----- 1 osni acts      4284 Aug 17 16:22 pdpttr_2.f
8 -rw----- 1 osni acts      361  Aug 17 16:22 pdpttr_2.i1
8 -rw----- 1 osni acts      4600 Aug 17 16:22 pdttr_col_major.c
8 -rw----- 1 osni acts      389  Aug 17 16:22 pdttr_col_major.i1
8 -rw----- 1 osni acts      4616 Aug 17 16:22 pdttr_row_major.c
8 -rw----- 1 osni acts      389  Aug 17 16:22 pdttr_row_major.i1
[ /usr/common/acts/SCALAPACK/hands-on/variou ]
```

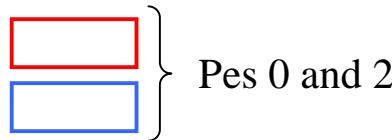
pddttrdrv.c (*pddttrdrv.f*): illustrates the use of the ScaLAPACK routines PDDTTRF and PDDTTRS to factor and solve a (diagonally dominant) tridiagonal system of linear equations $Tx = b$. After compilation, it can be executed with *llsubmit pddttrdrv.ll*.

pdpttr_2.c (*pdpttr_2.f*): illustrates the use of the ScaLAPACK routines PDPTTRF and PPPTTRS to factor and solve a symmetric positive definite tridiagonal system of linear equations $Tx = b$, in two distinct contexts. After compilation, it can be executed with *llsubmit pdpttr_2.ll*.

pdgesvddrv.f: reads a (full) matrix A from a file, distributes A among the available processors and then call the ScaLAPACK subroutine PDGESVD to compute the SVD of A , $A=USV^T$. It requires the file *pdgesvddrv.dat*, which should contain: line 1, the name of the file where A will be read from; line 2, the number of rows of A ; line 3: the number of columns of A . Considering the file *A.dat*:

- if $m=n=10$ the results are given in the file *A.SVD*
- if $m=10, n=7$: $\text{diag}(S)=[4.4926 \ 1.4499 \ 0.8547 \ 0.8454 \ 0.6938 \ 0.4332 \ 0.2304]$
- if $m=7, n=10$: $\text{diag}(S)=[4.5096 \ 1.1333 \ 1.0569 \ 0.8394 \ 0.8108 \ 0.5405 \ 0.2470]$

Data distribution for *pdpttr_2.c* (*pdpttr_2.f*)



| | | | |
|--------|--------|--------|--|
| 1.8180 | 0.8385 | | |
| 0.8385 | 1.6602 | 0.5681 | |
| 0.5681 | 1.3420 | 0.3704 | |
| 0.3704 | 1.2897 | 0.7027 | |
| 0.7027 | 1.3412 | 0.5466 | |
| 0.5466 | 1.5341 | 0.4449 | |
| 0.4449 | 1.7271 | 0.6946 | |
| 0.6946 | 1.3093 | | |

$$\begin{matrix}
 & & 0 & 2 & 0 & 2 \\
 & & x_1^{(1)} & x_1^{(2)} & x_1 & x_1 \\
 & & x_2^{(1)} & x_2^{(2)} & x_2 & x_2 \\
 & & x_3^{(1)} & x_3^{(2)} & x_3 & x_3 \\
 & & x_4^{(1)} & x_4^{(2)} & x_4 & x_4 \\
 & & x_5^{(1)} & x_5^{(2)} & x_5 & x_5 \\
 & & x_6^{(1)} & x_6^{(2)} & x_6 & x_6 \\
 & & x_7^{(1)} & x_7^{(2)} & x_7 & x_7 \\
 & & x_8^{(1)} & x_8^{(2)} & x_8 & x_8 \\
 & & 1 & 3 & 1 & 3
 \end{matrix} = \left\{ \begin{array}{cc|cc}
 & & 1 & 8 \\
 & & 2 & 7 \\
 & & 3 & 6 \\
 & & 4 & 5 \\
 & & 5 & 4 \\
 & & 6 & 3 \\
 & & 7 & 2 \\
 & & 8 & 1
 \end{array} \right\}$$

```

/*
***** This program illustrates the use of the ScalAPACK routines PDPTTRF ****
/* and PPPTTRS to factor and solve a symmetric positive definite */
/* tridiagonal system of linear equations, i.e., T*x = b, with */
/* different data in two distinct contexts. */
***** */

/* a bunch of things omitted for the sake of space */

main()
{
    /* Start BLACS */
    Cblacs_pinfo( &mype, &npe );
    Cblacs_get( 0, 0, &context );
    Cblacs_gridinit( &context, "R", 1, npe );
    /* Processes 0 and 2 contain d(1:4) and e(1:4) */
    /* Processes 1 and 3 contain d(5:8) and e(5:8) */
    if      ( mype == 0 || mype == 2 ){
        d[0]=1.8180; d[1]=1.6602; d[2]=1.3420; d[3]=1.2897;
        e[0]=0.8385; e[1]=0.5681; e[2]=0.3704; e[3]=0.7027;
    }
    else if ( mype == 1 || mype == 3 ){
        d[0]=1.3412; d[1]=1.5341; d[2]=1.7271; d[3]=1.3093;
        e[0]=0.5466; e[1]=0.4449; e[2]=0.6946; e[3]=0.0000;
    }
    if ( mype == 0 || mype == 1 ) {
        /* New context for processes 0 and 1 */
        map[0]=0; map[1]=1;
        Cblacs_get( context, 10, &context_1 );
        Cblacs_gridmap( &context_1, map, 1, 1, 2 );
        /* Right-hand side is set to b = [ 1 2 3 4 5 6 7 8 ] */
        if      ( mype == 0 ) {
            b[0]=1.0; b[1]=2.0; b[2]=3.0; b[3]=4.0;
        }
        else if ( mype == 1 ) {
            b[0]=5.0; b[1]=6.0; b[2]=7.0; b[3]=8.0;
        }
        /* Array descriptor for A (D and E) */
        desca[0]=501; desca[1]=context_1; desca[2]=n; desca[3]=nb;
        desca[4]=0; desca[5]=lda; desca[6]=0;
        /* Array descriptor for B */
        descb[0]=502; descb[1]=context_1; descb[2]=n; descb[3]=nb;
        descb[4]=0; descb[5]=ldb; descb[6]=0;
        /* Factorization */
        pdpttrf( &n, d, e, &ja, desca, af, &laf,
                  work, &lwork, &info );
        /* Solution */
        pdpttrs( &n, &nrhs, d, e, &ja, desca, b, &ib, descb,
                  af, &laf, work, &lwork, &info );
        printf( "MYPE=%i: x[:]= %7.4f %7.4f %7.4f %7.4f\n",
                mype, b[0], b[1], b[2], b[3]);
    }
}

```

pdppttr_2.c (pdpttr_2.f)

```

else {
    /* New context for processes 0 and 1 */
    map[0]=2; map[1]=3;
    Cblacs_get( context, 10, &context_2 );
    Cblacs_gridmap( &context_2, map, 1, 1, 2 );
    /* Right-hand side is set to b = [ 8 7 6 5 4 3 2 1 ] */
    if      ( mype == 2 ) {
        b[0]=8.0; b[1]=7.0; b[2]=6.0; b[3]=5.0;
    }
    else if ( mype == 3 ) {
        b[0]=4.0; b[1]=3.0; b[2]=2.0; b[3]=1.0;
    }
    /* Array descriptor for A (D and E) */
    desca[0]=501; desca[1]=context_2; desca[2]=n; desca[3]=nb;
    desca[4]=0; desca[5]=lda; desca[6]=0;
    /* Array descriptor for B */
    descb[0]=502; descb[1]=context_2; descb[2]=n; descb[3]=nb;
    descb[4]=0; descb[5]=ldb; descb[6]=0;
    /* Factorization */
    pdpttrf( &n, d, e, &ja, desca, af, &laf,
              work, &lwork, &info );
    /* Solution */
    pdpttrs( &n, &nrhs, d, e, &ja, desca, b, &ib, descb,
              af, &laf, work, &lwork, &info );
    printf( "MYPE=%i: x[:]= %7.4f %7.4f %7.4f %7.4f\n",
            mype, b[0], b[1], b[2], b[3]);
}
Cblacs_gridexit( context );
Cblacs_exit( 0 );
}

```

Using Matlab notation:

$T = \text{diag}(D) + \text{diag}(E, -1) + \text{diag}(E, 1)$

where

$D = [1.8180 \ 1.6602 \ 1.3420 \ 1.2897 \ 1.3412 \ 1.5341 \ 1.7271 \ 1.3093]$
 $E = [0.8385 \ 0.5681 \ 0.3704 \ 0.7027 \ 0.5466 \ 0.4449 \ 0.6946]$

Then, solving $T^*x = b$,

$\text{if } b = [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8]$
 $x = [0.3002 \ 0.5417 \ 1.4942 \ 1.8546 \ 1.5008 \ 3.0806 \ 1.0197 \ 5.5692]$

$\text{if } b = [8 \ 7 \ 6 \ 5 \ 4 \ 3 \ 2 \ 1]$
 $x = [3.9036 \ 1.0772 \ 3.4122 \ 2.1837 \ 1.3090 \ 1.2988 \ 0.6563 \ 0.4156]$

Block Cyclic Distribution

Consider the 12-by-10 matrix:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} & a_{1,8} & a_{1,9} & a_{1,10} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} & a_{2,7} & a_{2,8} & a_{2,9} & a_{2,10} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} & a_{3,7} & a_{3,8} & a_{3,9} & a_{3,10} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} & a_{4,7} & a_{4,8} & a_{4,9} & a_{4,10} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} & a_{5,7} & a_{5,8} & a_{5,9} & a_{5,10} \\ a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} & a_{6,5} & a_{6,6} & a_{6,7} & a_{6,8} & a_{6,9} & a_{6,10} \\ a_{7,1} & a_{7,2} & a_{7,3} & a_{7,4} & a_{7,5} & a_{7,6} & a_{7,7} & a_{7,8} & a_{7,9} & a_{7,10} \\ a_{8,1} & a_{8,2} & a_{8,3} & a_{8,4} & a_{8,5} & a_{8,6} & a_{8,7} & a_{8,8} & a_{8,9} & a_{8,10} \\ a_{9,1} & a_{9,2} & a_{9,3} & a_{9,4} & a_{9,5} & a_{9,6} & a_{9,7} & a_{9,8} & a_{9,9} & a_{9,10} \\ a_{10,1} & a_{10,2} & a_{10,3} & a_{10,4} & a_{10,5} & a_{10,6} & a_{10,7} & a_{10,8} & a_{10,9} & a_{10,10} \\ a_{11,1} & a_{11,2} & a_{11,3} & a_{11,4} & a_{11,5} & a_{11,6} & a_{11,7} & a_{11,8} & a_{11,9} & a_{11,10} \\ a_{12,1} & a_{12,2} & a_{12,3} & a_{12,4} & a_{12,5} & a_{12,6} & a_{12,7} & a_{12,8} & a_{12,9} & a_{12,10} \end{bmatrix}$$

Do the following block cyclic distributions:

- 3-by-3 blocking on a 3-by-2 process grid
- 4-by-4 blocking on a 2-by-3 process grid

Use <http://acts.nersc.gov/scalapack/hands-on/datadist.html> to compare